An Improvement of IP Address Lookup based on Rule Filter Analysis

An Improvement of IP Address Lookup based on Rule Filter Analysis

K. Guerra Pérez
kguerraperez01@qub.ac.uk.

X. Yang
x.yang@ecit.qub.ac.uk.

S. Sezer
s.sezer@ecit.qub.ac.uk.

The Institute of Electronics, Communications and Information Technology (ECIT)
Queen's University Belfast
Belfast, UK

Abstract— **Multi-bit trie is a popular approach performing the longest prefix matching for packet classification. However, it requires a long lookup time and inefficiently consumes memory space.**

This paper presents an in-depth study of different variations of multi-bit trie for IP address lookup. Our main aim is to study a method of data structure which reduces memory space. The proposed approach has been implemented using the label method in two approaches. Both methods present better results regarding lookup speed, update time and memory bit consumptions.

Keywords—Packet Classification; IP lookup; multi-bit trie algorithms; rule filter; Longest Prefix Matching

I. INTRODUCTION

Packet classification is a key function of network processing in a wide range of applications (e.g. a router/switch). Packet Classification has moved beyond the basic traditional network technologies, such as Ethernet switches or Multiprotocol Label Switching (MPLS) to complex level and is being promoted as the basis for Software-Defined Networking (SDN) and the OpenFlow protocol.

The applications of the next generation network require intensive design tasks on time/space complexity, a very large number of rules, high speed, scalability, flexibility, etc.

In general, the most common Ethernet Frame format used for Packet Classification includes amongst others, the following fields: *Source and Destination Port fields*, *Source and Destination IP Address fields* and *Protocol field* from a packet header. Longest-prefix Matching (LPM) is a common approach used for IP address lookup. It is a special case of Wildcard Matching that selects the entry in the prefix table with the greatest number of match bits.

In order to operate lookup function for internet traffic at line rates of 40Gbps and beyond, individual searches on each header field become necessary. In such cases, IP address field lookup becomes the bottleneck in terms of its length and the presence of wildcard.

According to Internet Protocol version 4 (IPv4), IP address fields -source and destination- contain 64 bits and its classification rules are formed by 128 bits, while IPv6 presents 256 bits for IP address source and destination fields and 512 bits for rules.

Taking into account this challenge and the fact that the number of entries in the flow tables is increasing exponentially [1], an algorithm for IP lookup with efficient update and lookup time is necessary for current Network applications and requirements.

The rest of the paper is organized as follows. In section II, we introduce the background and the related works. A filter set is analyzed in section III. In section IV, we present different approaches using a trie algorithm. Section V discusses the performance evaluation results using different filters and databases. We present a solution to improve the lookup performance. In section VII we compare with other structures. Finally, in section VIII, we conclude the paper.

II. RELATED WORK

Several Packet Classification solutions have been proposed for IP address lookup for many years. Nowadays, the Packet Classification problem is still a key for new Network applications and platforms, such as SDN.

Different tree/trie structures based on *Search on Length Tree* are considered as alternative methods to support Wildcard Matching. Two groups can be categorized in this kind of structures; Binary tree-based and Multi-bit trie-based structures.

Binary Search Tree algorithms [3][4] use each data bit in order to know the next child node of the next level. This method requires higher latency and more storage with a larger address width.

Some algorithms based on binary search are presented, such as Practical Algorithm to Retrieve Information Coded in Alphanumeric (PATRICIA) [5], which compresses each chain to a single node and the full lookup is not necessary. A PATRICIA tree loses information while compressing chains and the lookup complexity is high and it does not support LPM.

Path Compressed Trie [6] reduces space requirements as well as lookup time required by PATRICIA.

Tree structures present inefficient memory storage. H. Park et. al. [7] proposes a method to reduce the number of empty nodes. However this method is applied to balanced binary trie.

V. Srininvasan et. al. [8] presented Grid of Trie (GoT), is based on a binary branching trie of tries optimal for two fields. The incremental update is difficult in this method and even the later versions [9] .

Multi-bit Trie algorithms [10] examine a group of bits at the same time. Multi-bit tries still do linear search on lengths, but since the trie is traversed in larger strides the search is faster. This method reduces the depth of the trie and it is an easy hardware solution mapped into pipeline stages. One of the main disadvantages is the need to store children nodes for each new created node, denoting an inefficient memory usage. The branches of Multi-bit Tries in each level represent a fixed size prefix and, consequently it is not flexible for prefixes of different lengths. Multi-bit is traversed from root until the leaf node is reached.

LC-trie [11] is a trie structure with combined path compression and level compression to reduce the number of nodes, but it is not suitable for a large number of entries and it does not support incremental update.

Lulea [12] reduces storage consumption but its benefits depend on the structure and it does not support incremental update.

Variable-Stride Multi-bit Trie [13] presents a multi-bit trie with variable and fixed-stride capacity but the memory requirement is worse than the other algorithms.

Multi-prefix trie (2-MPT) [14] reduces the number of lookup memory accesses. This method stores extra prefix information in each node, sacrificing memory space.

Other approaches for IP lookup based on *Search on Value* do not support LPM and have the need of extra phases to convert from prefix to range data.

III. RULE FILTER ANALISYS

A rule is composed of five or more fields and it defines an action. When an input packet matches against a rule, the corresponding action is applied to the input packet. A set of determined rules is called a filter.

Rule syntaxes are widely researched. Rules present certain patterns that can be explored by algorithms. For example, on one hand, trie-based algorithms build the structure according to the rule prefixes. On the other hand, Distributed Cross-producting Field Label (DCFL) [15] labels the unique rule fields. DCFL applies labels into multi-bit trie algorithms for a lookup process instead of rules.

Consequently, we examined different kinds of filters: Accesses Control List (ACL), Firewall (FW) and IP Chain (IPC), with different sizes [17]. The size of the given rule filters is summarized in Table I and are named 1 K, 5 K and 10 K rules in order to simplify the denomination of rule sets.

TABLE I. NUMBER OF RULES OF THE DIFFERENT FILTER SETS

Filters	1 K rules	5 K rules	10 K rules
ACL	916	4415	9603
FW	791	4653	9311
IPC	938	4460	9037

As an example, Table II shows the statistic results concerning the number of unique rules for each dimension extracted from the worst case filter of 10K rules.

This analysis reveals that there exists a rule field repetition which offers design space for improvement on storage capacity, lookup time, incremental update time, etc.

TABLE II. ANALYSIS OF RULE FILTERS

Maximum No. Unique Fields	ACL (9603 rules)	FW (9311 rules)	IP (9037 rules)
IP Address	4784	6951	2726
Port	108	43	54
Protocol	3	3	3

IV. LOOKUP APPROACHES AND IMPLEMENTATIONS

In this section, the goal is to study a new approach focused on the rule set survey, independently of the algorithm. This work studies IP address lookup which is the bottleneck in Packet Classification.

Much research has been performed on algorithms based on Trie/Tree structure support LPM. With the Multi-bit trie algorithm in particular, being extensively has been investigated due to its ability to improve both software and hardware platforms. Our objective is to investigate and compare the same data structure with three different approaches. In our work, three Multi-bit trie implementations have been performed under the same conditions using different size ACL1, FW1 and IPC filters. For those filters, two 32-bit IP address fields, source and destination, from the headers are utilized.

The memory space required for Multi-bit trie nodes is $O(2^s)$ where s corresponds to the number of bit of strides. Moreover, conventional multi-bit trie presents disadvantages of building rule filters with large prefix size. Bearing this in mind, we divide the IP address fields into smaller segments. For example, 16-bit prefix segments can be divided into four tries with fixed number of bits of each trie. Afterward, we apply a multi-bit trie algorithm for each independent segment in parallel. In this section, the results are analyzed for the worst case from the independent search of the different experiments.

The performance evaluation of the software-based algorithms is performed according to certain standards [2]. The lookup and update speeds are evaluated by the worst case number of memory accesses. In Packet Classification the IP lookup using trie algorithms not only depends on the trie depth but also the highest priority matching rule search. Incremental update is essential for the current requirements. Finally, the memory space is a key metric for Packet Classification where the trie node information must be kept as well as the rule set. Due to the recent growth of Internet traffic, a large amount of entries is essential for Packet Classification in current networks.

A. Original Multi-bit Search Trie

Each node of the original Multi-bit Search Trie represents a determined n-bits prefix in the trie algorithm. Each leaf node stores a list of rules and the highest priority matching rule

(HPMR) is found using a simple linear search. Using this methodology, it is expected that memory space as well as long lookup time will be inefficient due to the list of rules stored in each trie node. However, supposing there are no repeated rules, this experiment runs at a fast insertion process.

Different scenarios are studied for IPv4 using tries with four levels per dimension, in order to acquire the optimal parameters values. Table III shows an example using source IP address fields.

TABLE III. EXAMPLE OF RULE FILTER

Rule Filter	Source IP address	Hexadecimal
R0	192.145.181.80/29	C0.91.B5.50
R1	192.145.181.80/32	C0.91.B5.50
R2	192.145.181.84/29	C0.91.B5.54
R3	192.145.180.00/24	C0.91.B4.00

B. Experiment 1: Multi-bit Trie with labeled rule fields

Experiment 1 (EXP_1) is based on an improved structure of the original Multi-bit Search trie algorithm. According to the rule filter analysis, EXP_1 performs the lookup process using the label method [15]. This method is motivated by the rule analysis presented in Table II, which demonstrates that the number of unique rules is lower than the total number of rules. Thus, the label represents all rules containing this field. The main idea of this work is to label each unique rule field. By storing the labels instead of the entire rule information, memory consumption can be significantly reduced.

In our implementation, a label is assigned to the unique 16-bit partitions of each rule field that must be stored in the multi-bit tries. Consequently, each trie links with a certain label filter. The independent filter information is composed of a label and a counter in order to support incremental update.

The wildcard bits are taken into account as different labels. An example shown in Table IV covers R0 and R1 as different labels due to the different masks. On the contrary, R2 and R4 are named with the same label. In order to find the HPMR, the combinations of the labels are stored in a final label filter.

With this method, we expect that this experiment will require less memory storage than original Multi-bit trie. Furthermore, the lookup process is expected to be faster. However, the update processes can be compromised by the corresponding label lookup into the filters.

TABLE IV. LABEL ASSIGNEMENT

Partition Labels			
Higher 16-bits	**Label**	**Lower 16-bits**	**Label**
C0.91/16	A	B5.50/13	A
		B5.50/16	B
		B5.54/16	C
		B4.00/8	D

C. Experiment 2: Multi-bit Trie with labeled nodes

Experiment 2 (EXP_2) uses label method on a multi-bit trie. In this case, the trie nodes are labeled instead of the unique field.

After all search results are available from each trie, the final lookup is performed in another label filter with combinations of labels.

The experiment demonstrates not only a reduction of memory space, but also an improved lookup speed. Since leaf nodes do not contain any rule list, the goal of EXP_2 is also to avoid the linear search into the trie. Moreover, the corresponding label does not have to be searched through a filter beforehand. The label will be retrieved when the leaf node is reached.

V. PERFORAMCE EVALUATION

The experiment results from the different scenarios are presented and discussed in this section. As mentioned above, we discuss the three experiments in the four situations shown in Table V. All of them are constructed with 3-level multi-bit tries with diverse level distributions.

As previously mentioned, each IP address field is divided into two 16-bit segments to be analyzed in two multi-bit tries. The IP address lookup system is composed of four 3-level multi-bit tries; two for source address field and two for destination address field.

In this work, different trie distributions are explored in order to work with the optimal multi-bit trie structure. All trie nodes belonging to the same level have the same number of bits.

As shown in Table V, situation 1 has a 4-6-6 bit distribution. The trie structures are organized as 4-4-8 bits in situation 2. The levels are spread in 5-5-6 bits for each trie in situation 3 and finally, situation 4 works with 4-5-7 bits.

This survey analyses the main parameters for lookup process and update process performance according to the memory access requirements, and number of occupied bits. Three kinds of filters are used with three different rule-set sizes, at different packet databases.

Because all the experiments are based on a multi-bit search trie structure, the number of stored nodes and the number of memory accesses for the lookup process are the same values in each situation.

From Table V, situation 1 and situation 3 overcome the others in all parameters. Moreover situation 3 presents a slight improvement over situation 1. However, the first situation is adapted to the three experiments because it gives better result in a general evaluation.

A. Lookup Process

As mentioned in the previous section, the IP address lookup performs the same process in the three experiments.

Nevertheless, each experiment gives different results for the search on the highest priority matching rule. The analysis of HPMR lookup process is discussed in Section IV.

Because trie nodes in the original Multi-bit trie contain a list of rules, the lookup process needs to compare the rules contained in the four resulting lists until the common matching rule is found using a simple linear search. The number of memory accesses per rule for original Multi-bit trie

TABLE V. VALUES FOR ALL EXPERIMENTS IN EACH CASE

	Type of Filters	ACL			FW			IPC		
Situation 1	Filter size	1 K	5 K	10 K	1 K	5 K	10 K	1 K	5 K	10 K
	Mem. Acc. Trie Lookup	3.993	3.997	3.998	3.722	4.00	4.00	3.942	3.974	3.949
	Total Stored Nodes	13952	28928	66112	12160	215599	262144	22592	33344	65920
	Valid Stored Nodes	1140	3293	8480	1898	15031	36929	2287	5046	9241

	Type of Filters	ACL			FW			IPC		
Situation 2	Filter size	1 K	5 K	10 K	1 K	5 K	10 K	1 K	5 K	10 K
	Mem. Acc. Trie Lookup	3.993	3.998	3.999	3.854	4.00	4.00	3.942	3.974	3.986
	Total Stored Nodes	36624	72464	150032	30160	259856	262144	58576	80096	131392
	Valid Stored Nodes	946	33717	9106	1531	15837	38338	9542	27230	50546

	Type of Filters	ACL			FW			IPC		
Situation 3	Filter size	1 K	5 K	10 K	1 K	5 K	10 K	1 K	5 K	10 K
	Mem. Acc. Trie Lookup	3.993	3.996	3.998	3.615	4.00	4.00	3.942	3.974	3.949
	Total Stored Nodes	13824	28160	66080	11456	215648	262144	22240	32928	65984
	Valid Stored Nodes	1204	3229	8384	1340	13702	35691	2351	5110	9305

	Type of Filters	ACL			FW			IPC		
Situation 4	Filter size	1 K	5 K	10 K	1 K	5 K	10 K	1 K	5 K	10 K
	Mem. Acc. Trie Lookup	3.993	3.997	4.9981	3.723	4.00	4.00	3.942	3.974	3.949
	Total Stored Nodes	21984	45536	103008	18400	251296	262144	36320	51200	95424
	Valid Stored Nodes	948	2874	8241	1696	16324	38868	6721	17009	32185

is quite high, achieving 5.89 x10^7 memory accesses in the worst-case.

For this reason, the result shown in Fig. 1 is related to the average number of memory accesses required by EXP_1 and EXP_2 in the corresponding filters.

Fig. 1 reveals that EXP_2 performs the worst HPMR lookup due to this experiment using a unique very large label filter. This filter is traversed with linear search.

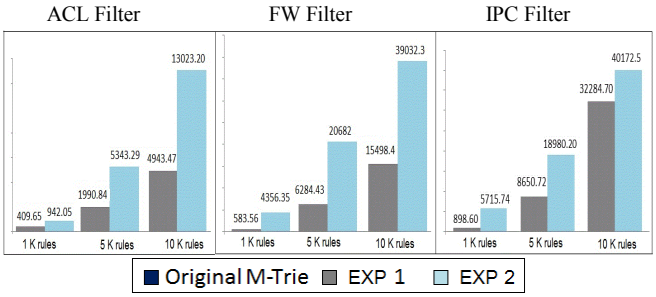

Fig. 1. Average number of memory accesses of Lookup process for EXP_1 and EXP_2

B. *Update Process*

Multi-bit trie supports incremental update and consequently, all experiments, which are based on this algorithm, are able to hold incremental update.

The results for the insertion process regarding the number of memory accesses are shown in Fig 2. This figure represents the average number of memory accesses per rule required for each experiment to insert a rule in the trie.

Because in the original Multi-bit trie and EXP_2 each rule is inserted into the trie, both experiments show the same results.

However in EXP_1, it is not necessary to insert each rule into the trie if the label of the input rule field is already stored in the label filter, contributing fewer memory accesses for insertion.

A rule is inserted immediately into the trie in original Multi-bit trie. On the contrary, an extra phase is required using EXP_1 and EXP_2 in order to lookup the label or to add the label in the label filter.

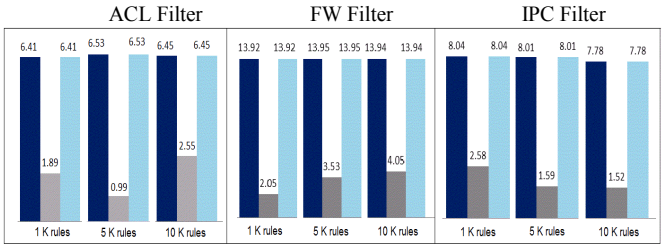

Fig. 2. Insertion Process in each experiment trie for each type of filter

In particular, in EXP_1 extra time is necessary to find the corresponding label. It is supposed that the independent label filters for each trie work in parallel in order to find the corresponding label. Afterward, the resulting labels from each trie are deposited in the final label filter as a combination.

All node labels are combined in EXP_2, including wildcards nodes, after each trie insertion and saves into the label filter. This experiment does not perform any search process in the label filter for the rule insertion.

The worst case of average number of memory accesses of label filter insertion is shown in Fig 3. The graphs prove that in EXP_1, larger insertion time is required due to the label pre-search in the independent filters.

Deletion process is examined, erasing 50, 100 and 150 rules in 1 K rules, 5 K rules and 10 K rules respectively. In this case, the results shown in Fig. 4 demonstrate that, in original Multi-bit trie, the rule must be deleted from all lists belonging to all leaf nodes found, including wildcard nodes, using a simple linear search.

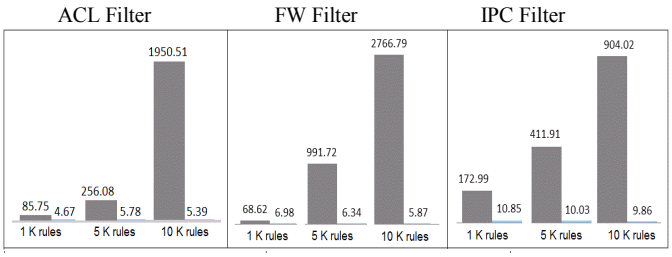

Fig. 3. Average number of memory accesses of Insertion Process for Label Filter of EXP_1 and EXP_2

Likewise, in EXP_1, the linear search is used to delete the label from a shorter label list of the leaf nodes but only if it is necessary. This corner case happens when the corresponding counter belonging to a label of the independent filters is set to zero. In EXP_2, the counter of the leaf nodes is simply decremented and deletes the node when this node counter is changed to zero.

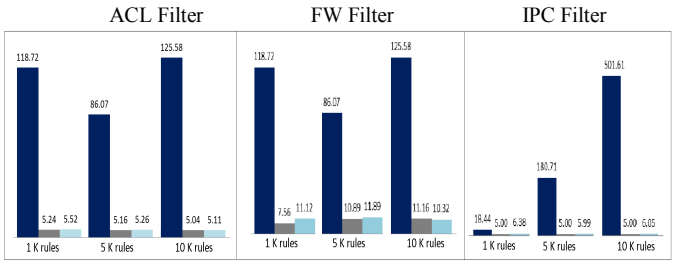

Fig. 4. Average number of memory accesses of Deletion Process in each experiment trie

However, the results are the opposite for the label filter searches performed in EXP_1 and EXP_2 according to Fig. 5. The time needed to find the rule is greater for the label filter in EXP_2. This outcome is due to the size of label filter of EXP_2 being much larger than all independent label filters and even the final label filter used in EXP_1. Even though any filter is needed with original Multi-bit trie, deletion is faster using EXP_1.

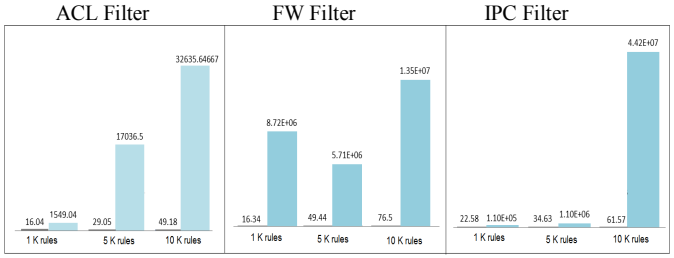

Fig. 5. Average number of memory accesses of Insertion Process for Label Filters of EXP_1 and EXP_2

C. *Memory Space*

The memory storage required by each experiment is discussed in Section C and shown in Fig. 6 using a diverse set of filters. The results show that the problem found in original Multi-bit trie is overcome in the two following experiments by including the label filters. Fig. 6 reveals that less memory storage is required in EXP_1.

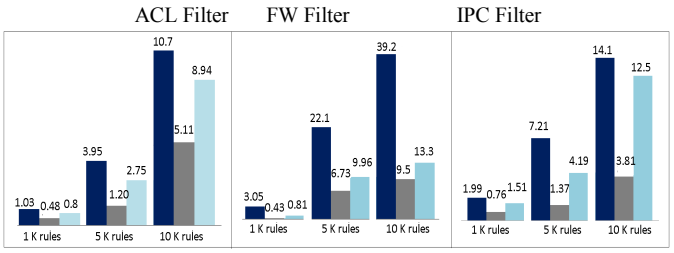

Fig. 6. Memory Space required of each experiment in Mbits.

The information stored in each experimental trie is shown in Table VII. This information is related to the number of stored rules in the Multi-bit trie and the number of stored labels in the EXP_1 tries. Likewise, the size of the labels filters used in EXP_1 and EXP_2 are shown in Table VII.

VI. IMPROVEMENT

As discussed in Section V, the methodology used in EXP_1 exceeds the rest of the experiments in terms of performance in the trie. EXP_1 also gives better results regarding the memory space required. However, in EXP_2 a fewer memory accesses is used for insertion and deletion processes. It is due to the need of a previous label search for both processes in EXP_1.

Considering the size of all filters in EXP_1, independent label filters for each trie and a label filter for the combinations, a hash table is used in order to reduce the lookup time. 6951 independent labels are needed in the worst case in EXP_1 for the 10 K rule set.

The filters require the same size but they include a list of collisions as unique difference, with the purpose of handling the possible collisions. The average number of collisions is two in all cases.

Consequently, the lookup time is reduced using the same system for IP address fields search. Table VI summarizes the number of memory accesses required for EXP_1 with a hash table included.

TABLE VI. IP LOOKUP PERFORMANCE USING HASH-FUNCTION

Avg. No.	Filters	1 K	5 K	10 K
Memory Accesses Insertion	**ACL**	2.14	2.24	2.33
	FW	2.03	1.15	1.56
	IPC	1.64	1.88	2.91
Memory Accesses Lookup	**ACL**	2.86	7.13	5.13
	FW	10.16	1.65	3.07
	IPC	5.59	53.03	24.38

Hash function can be applied to the Label Filter in EXP_2. However, this method does not affect on the multi-bit trie performance. However, our work overcomes DCFL by avoiding linear search of corresponding label.

VII. COMPARISON

The search performance of EXP_1 using hash function is similar to other algorithms such as 2-MPT with 25 memory accesses to search the HPMR in the worst-case.

TABLE VII. INFORMATION STORED AND LABEL FILTERS

Experiments	Type of Filters	ACL			FW			IPC		
	Filter size	1 K	5 K	10 K	1 K	5 K	10 K	1 K	5 K	10 K
Original Multi-bit trie	No. Stored Rules	6238	30469	64513	26183	155098	308160	13232	60683	118234
EXP_1	Label Filter sizes	1414	5999	15374	1138	10749	29689	1743	6181	12648
	No. Stored Labels	1271	30471	64513	2061	16259	39385	2887	6675	12734
EXP_2	Label Filter sizes	4281	25512	51806	5519	29521	54648	10177	44749	89139

The update process is very hard in algorithms such as LC-trie or Lulea. The update process presents high overhead in more recent multi-bit trie IP lookup algorithms, such as Trie Bitmap and DIR-24-8-BASIC [16],

This method obtains advantages regarding the update process. As in multi-bit schemes, the experiments support incremental update. Moreover, as stated in Section V, EXP_1 reduces the update time in comparison with other trie structures. The original Multi-bit Search trie and 2-MPT need 13.95 and 16 memory accesses respectively against 5.87 memory accesses for EXP_1 in the worst-case. That is because it is not necessary to go through the tries for every insertion or deletion using label method as in EXP_1.

Different from binary-based trees or GoT, a multi-bit trie has a static size, where the number of maximum nodes is known and has a determined depth. Binary trie can achieve 32 levels for IPv4 and LC-Trie can reach 14 of trie depth, FST can have 7 heights and 2-MPT contains 13 levels. All of them are overcome by any of three cases studied where the maximum trie height is three levels using 16-bit partition and work in parallel.

Furthermore, the duplicated rules are avoided in all tries as happens with most trie-based algorithms in EXP_1 and EXP_2. Moreover the replication of the labels within a trie is avoided in EXP_2.

In EXP_1, empty nodes are stored, resulting in moderate memory inefficiency. Despite of these disadvantages, less memory storage is required in EXP_1 than that of original Multi-bit trie or EXP_2, including the label filters. According to the number of the nodes, Multi-bit trie structures waste memory space with empty nodes. In our experiment, the maximum number of the stored nodes is 262144 with less than 15% of them containing valid information. It is solved using path-compressed binary trie or 2-MPT where the empty nodes are replaced by valid nodes.

VIII. CONCLUSION

Packet classification requires multiple field lookups on the packet header. IP address fields require a major dedication due to the large field size and the difficulty to find the matching rule with wildcard. The contribution of this work is summarized in four main goals. Firstly, this paper presents an evaluation of multi-bit tries in obtaining high performance. Optimal distribution parameters for a fixed 3-level trie are suggested to implement the proposed solution. Secondly, a survey of different rule filters has been performed, which is critical for our proposed method. Thirdly, the Multi-bit trie algorithm with the best parameters has been implemented in order to support LPM using different approaches. Both approaches obtain better performance than the original multi-bit trie. Finally, EXP_1, with unique rule field labeling and hash table lookup, has been proved to be a better solution for LPM.

This method can be applied to others algorithms with the same structure and even combined with other methods. Our proposed solution is straightforward implementable into hardware platforms and is applicable to IPv6 format.

REFERENCES

[1] P. J. B. King and N. K. Vlachos " Internet Traffic Classification and Features: Current Levels and Future Projections" , *PG Net* , June 2013

[2] H. Park, H. Hong, S. Kang, "An Efficient IP address lookup algorithm based on a small balanced tree using entry reduction", *Computer Network,* pp. 231-243, 2012.

[3] P. Gupta and N. Mckeown, "Dynamic Algorithms with Worst-case performance Packet Classification", *NETWORKING*, pp. 528-539, 2000.

[4] M. de Berg, O. Cheong, M. van Kreveld, M. Overmars "Computational Geometric: algorithms and applications" (2008), 3rd edition.

[5] D. R.Morrison, "PATRICIA: Practical Algorithm To Retrieve Information Coded in Alphanumeric", *J ACM,* Vol 15, pp. 514-534, October 1968.

[6] P. Gupta and N. Mckeown, "Algorithms for packet classification", in *IEEE Network*, vol. 5, pp. 24-32, 2001.

[7] H. Park, H. Hong, S. Kang "An efficient IP address Lookup algorithm based on a small balanced tree using entry reduction", *The International Jornal of Computer and Telecommunications Networking,* Vol 56 pp. 231-243, Janyary 2012.

[8] V. Srinivasan, S. Suri, G. Varghese abd M. Waldvogel. "Fast and Scalable Layer four Switching," *ACM Sigcomm*, Vol. 28, pp. 191-202, October 1998.

[9] Y. Chang, Y. Lin, C. Lin "Grid of Segment Trees for Packet Classification". *IEEE AINA*, pp. 1144-1149, 2010.

[10] M. A. Ruiz-Sanchez, E.W Biersack and W. Dabbous, "Survey and Taxonomy of IP Address Lookup Algorithms", *IEEE The Magazine of Global Internetworking*, Vol. 15, pp. 8-23, March 2001.

[11] S. Nilsson and G. Karlsson, "IP-address lookup using LC tries,". *IEEE Journal on Selected Areas in Communications*, Vol. 17, pp. 1083-1092, June 1999.

[12] M. Degermark, A. Brodnik, S. Carlsson and S. Pink. "Small forwarding tables for fast routing lookups," *ACM Sigcomm*, pp. 3-14, October 1997.

[13] S. Sahni, K. S. Kim, "Efficient Construction of Variable-Stride Multi-bit Tries For IP Lookup". *IEEE SAINT,* 2002.

[14] S. Hsieh, Y. Huang, Y. Yang. "Multiprefix Trie: A New Data Structure for Designing Dynamic Router-Tables" *IEEE Transaction on Computer,* pp. 693-706, May 2011.

[15] D. E. Taylor and J.S. Turner, "Scalable Packet Classification using Distributed Crossproducting of Field labels", *IEEE INFOCOM 2005*, Vol. 1, pp.269-280, March 2005.

[16] P. He, H. Guan, G. Xie, K. Salamatian "Evaluating and Optimizing IP Lookup on Many core Processors*". ICCCN 2012*, pp. 1-7, 2012

[17] H. Song, www.arl.wustl.edu/~hs1/PClassEval#3._Filter_Sets, accessed on 7th January 2014